Пермский филиал федерального государственного автономного образовательного учреждения высшего образования «Национальный исследовательский университет

«Высшая школа экономики»

Факультет экономики, менеджмента и бизнес-информатики

Образовательная программа бакалавриата «Программная инженерия»

**ОТЧЕТ  
по учебной практике**

Выполнил студент группы ПИ-19-1 Мамедов Нурлан Рагим оглы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Проверил:

Руководитель практики   
доцент кафедры информационных

технологий в бизнесе

Бузмаков Алексей Владимирович

\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка) (подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

Пермь, 2020 год

# Аннотация

В данном отчете описаны этапы проектирования, разработки и тестирования 12 задач. Программы реализованы на языке высокого уровня C#.

Работа выполнена студентом НИУ ВШЭ Пермь Мамедовым Нурланом Рагим оглы. Кафедра информационных технологий в бизнесе.

Работа содержит 41 страниц формата А4 основного текста, включая 12 глав соответствующей каждой учебной задаче.

Библиографический список включает в себя 4 источника.

В каждой главе содержится анализ задачи (установка функциональных и не функциональных требований к задаче), проектирование алгоритмов и тестирования ПО. Приложения к задачам содержат реализацию задач.

# Введение

Главной задачей данной работы является закрепление и развитие практических навыков в области программирования, алгоритмизации при помощи современных средств разработки, а также в области аналитической и проектной деятельности.

Целью данной работы является закрепление знаний и навыков, полученных по дисциплинам «Дискретная математика», «Компьютерный практикум по основам алгоритмизации и методам программирования», «Линейная алгебра», «Программирование», «Введение в программную инженерию», а также приобретение опыта алгоритмизации задач.

Для достижения поставленной цели необходимо:

1. Наличие практических навыков работы с языком программирования высокого уровня C# и современных сред разработки для реализации предложенных программ.
2. Навык работы с системой контроля версий посредством использования веб-сервиса GitHub.
3. Проведение тестирования разработанных систем и алгоритмов по критериям черного ящика.
4. Развитие практических навыков применения объектно-ориентированного программирования.

Основная часть данной работы содержит результаты выполнения 12 заданий учебной практики и включает в себя: постановку задачи, формат входных и выходных данных, описание решения задачи, разработка функциональных и нефункциональных требований, разработку алгоритмов, процесса реализации и результатов тестирования программы.

# Задача 1

Постановка задачи:

*«Вы работаете в фирме, занимающейся разработкой компьютерных игр. Сейчас вы занимаетесь разработкой новой компьютерной игры "Атака летающих тарелок". По сюжету игры на планету Зумла приземляются летающие тарелки, и их надо уничтожать. Игрок управляет лазерной пушкой. Для того, чтобы произвести выстрел он указывает две точки на поверхности Зумлы (которая в игре считается плоской), через которые должен проходить лазерный луч (который является прямой).*

*Вы должны написать программу, определяющую, какие летающие тарелки были уничтожены выстрелом»*

Входные данные: Первая строка входного файла INPUT.TXT содержит целое число n (1 ≤ n ≤ 30000) - число приземлившихся летающих тарелок. Вторая строка содержит числа xp1, yp1, xp2, yp2 - координаты точек, через которые проходит лазерный луч. Далее идут n строк, каждая из которых содержит описание одной летающей тарелки в формате xi yi ri, где xi, yi - координаты центра, ri - радиус тарелки. Все числа целые и не превосходят по модулю 10000. Радиусы летающих тарелок - целые и положительные. Летающие тарелки могут касаться и пересекаться.

Выходные данные: В первую строку выходного файла OUTPUT.TXT выведите количество уничтоженных летающих тарелок. Во вторую строку выведите номера уничтоженных летающих тарелок в возрастающем порядке. Тарелка считается уничтоженной, если она имеет, хотя бы одну общую точку с лазерным лучом.

Программа должна выполняться не более чем за 1 секунду и требовать не более чем 16 МБ памяти.

Необходимо написать программу, которая высчитывала бы перекрытие «лазера» (прямой) и «летающих тарелок» окружностей на плоскости. Заданы количество окружностей, две точки, через которые проходит прямая, точки центров окружностей и соответствующие радиусы.

## Анализ

Для решения задачи нужно:

1. Составить уравнение прямой (лазера) по двум точкам в виде:

(1.1)

Где A, B, C выводятся из формулы уравнения прямой, проходящей через две заданные точки на плоскости:

(1.2)

Тогда:

(1.3)

1. Расстояние от точки до прямой равно:

(1.4)

1. Теперь требуется пройти циклом по всем окружностям и записывать в ответ те, у которых радиус меньше или равен расстоянию от точки до прямой.

## Проектирование

Ниже представлена блок-схема, составленная на основе вышеописанного анализа и приведенного алгоритма.

Завершение

Нет

Да

Записывает тарелку в результат

Если |A \* x + B \* y + C| / Math.Sqrt(A \* A + B \* B) меньше радиуса тарелки

Проходит по всем координатам тарелок

A = y2 – y1

B = x1 – x2

C = x2 \* y1 – x1 \* y2

Считывание количества тарелок, координат лазера, координат центра тарелок и их радиуса

Начало

## Тестирование



Рисунок 1.1 Тестирование задачи 1

Требования к скорости работы и используемой памяти были выполнены.

# Задача 2

Петя разгадывает головоломку, которая устроена следующим образом. Дана квадратная таблица размера N×N, в каждой клетке которой записана какая-нибудь английская буква. Кроме того, дан список ключевых слов. Пете нужно, взяв очередное ключевое слово, найти его в таблице. То есть найти в таблице все буквы этого слова, причем они должны быть расположены так, чтобы клетка, в которой расположена каждая последующая буква слова, была соседней с клеткой, в которой записана предыдущая буква (клетки называются соседними, если они имеют общую сторону — то есть соседствуют по вертикали или по горизонтали). Например, на рисунке показано, как может быть расположено в таблице слово olympiad.

Когда Петя находит слово, он вычеркивает его из таблицы. Использовать уже вычеркнутые буквы в других ключевых словах нельзя. После того, как найдены и вычеркнуты все ключевые слова, в таблице остаются еще несколько букв, из которых Петя должен составить слово, зашифрованное в головоломке.

Помогите Пете в решении этой головоломки, написав программу, которая по данной таблице и списку ключевых слов выпишет, из каких букв Петя должен сложить слово, то есть какие буквы останутся в таблице после вычеркивания ключевых слов.

Входные данные: Во входном файле INPUT.TXT записаны два числа N (1 ≤ N ≤ 10) и M (0 ≤ M ≤ 100). Следующие N строк по N заглавных английских букв описывают ребус. Следующие M строк содержат слова. Слова состоят только из заглавных английских букв, каждое слово имеет длину от 1 до 100 символов. Гарантируется, что в таблице можно найти и вычеркнуть по описанным выше правилам все ключевые слова.

Выходные данные: В выходной файл OUTPUT.TXT выведите в алфавитном порядке оставшиеся в таблице буквы.

Программа должна выполняться не более чем за 1 секунду и требовать не более чем 16 МБ памяти.

## Анализ

Изначально может показаться, что для решения задачи, придётся реализовывать весь алгоритм вычёркивания слов из таблицы. Но при тщательном чтении обнаруживается, что порядок букв в ответе совершенно неважен. Отталкиваясь от этого, для решения задачи можно составить следующий алгоритм:

1. Считать входные данные.
2. Записать все слова таблицы в одну строку.
3. Записать все ключи в одну строку.
4. Пройти по каждому символу ключей.
5. Удалять вхождения символа ключей в строку слов.
6. Отсортировать оставшиеся буквы в строке слов в алфавитном порядке.

## Проектирование

По вышеописанному алгоритму составлена данная блок-схема.

Начало

Считывание входных данных

Запись всех слов таблицы в одну строку

Запись всех ключей в одну строку

Есть ли символ в строке слов таблицы?

Нет

А

Вывод ответа

Завершение

Проходит по каждому символу строки ключей

Да

А

Удаление символа из строки слов таблицы

## Тестирование



Рисунок 2.1 Тестирование задачи 2

Требования к скорости работы и используемой памяти были выполнены.

# Задача 3

Даны действительные числа x, y. Определить, принадлежит ли точка с координатами x, y заштрихованной части плоскости (см. рис. 1.3).

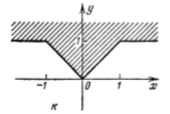


Рисунок 3.1 Заштрихованная часть плоскости

## Анализ

Задача являет собой проверку на принадлежность точки графикам:

(3.1)

Для решения задачи необходимо:

1. Считать координаты точки.
2. Проверить на соответствие условию.
3. Вывести результат.

Местом потенциальной ошибки является ввод координатов. Поэтому требуется дополнительно написать метод для проверки ввода.

## Проектирование

Согласно проведенному анализу, были составлены блок-схемы для методов.

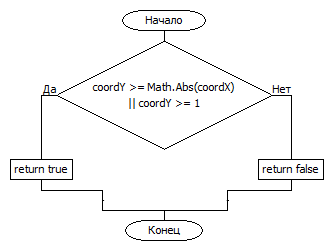


Рисунок 3.2 Блок-схема для задачи 3

Дополнительно была создана перегрузка метода для проверки принадлежности точки с одной координатой (координата по оси абсцисс принимается за нуль).

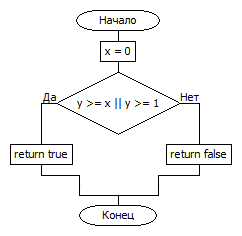


Рисунок 3.3 Блок-схема проверки на принадлежность с одним параметром

Также был спроектирован метод для проверки ввода пользователя.

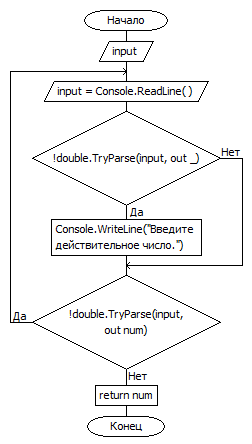


Рисунок 3.4 Блок-схема проверки ввода пользователя

## Тестирование

Для тестирования был составлена таблица тестов. Таблица содержит входные данные, ожидаемые выходное данные, критерии покрытия.

Таблица 3.1 Таблица тестов для задачи 3

|  |  |  |  |
| --- | --- | --- | --- |
| **Номер теста** | **Входные данные** | **Ожидаемые выходные данные** | **Примечание** |
| 1 | x = 0 y = 0 | Точка (0, 0) принадлежит графику | 1.1, 2.1, 3.1, 3.3 |
| 2 | x = 1 y = 1 | Точка (1, 1) принадлежит графику | 1.2, 2.2, 3.1, 3.3 |
| 3 | x = 2 y = 0 | Точка (2, 0) не принадлежит графику | 1.3, 2.1, 3.2 |
| 4 | x = -1 y = 2 | Точка (-2, 2) принадлежит графику | 2.3, 3.1 |

Тесты в таблице покрывают критерии чёрного ящика в таблице НАЗВАНИЕ ТАБЛИЦЫ И ПРИЛОЖЕНИЯ

# Задача 4

Дано действительное положительное число ε. Методом хорд вычислить с точностью ε\*) корень уравнения f(x) = 0.

## Анализ

Для решения задачи необходимо поочередно находить значения последовательности по формуле (1.6) в пределах заданных границ, пока не будет достигнута заданная точность. Для измерения точности был принят модуль разницы между текущим и предыдущим элементом, если меньше заданной точности, то останавливает выполнение программы.

(4.1)

Входные данные: исходная функция и границы вычисления значения.

Исходная функция по условию задачи равна:

(4.2)

Тогда её производная:

(4.3)

Выходные данные: действительное число, вычисленное с заданной точностью.

Также стоит учесть проверку ввода пользователя на положительные действительные числа.

## Проектирование

Согласно проведенному анализу и алгоритму, составленному на его основе, была спроектирована следующая блок-схема (см. рис. 4.1)

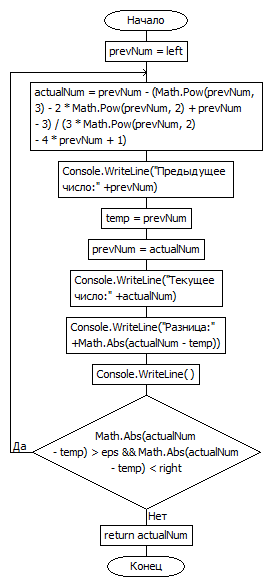


Рисунок 4.1 Блок-схема для задачи 4

Для метода проверки ввода пользователя была составлена блок-схема, представленная на рис. 4.2.

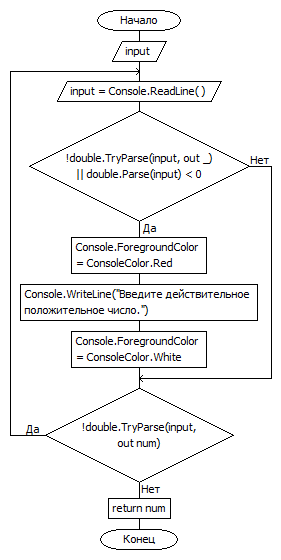


Рисунок 4.2 Блок-схема проверки метода пользователя

Также дополнительно был создан интерфейс для программы с использованием платформы WPF.

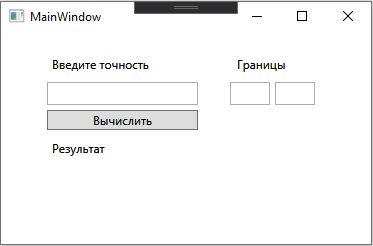


Рисунок 4.3 Интерфейс для программы для задачи 4

## Тестирование

Для тестирования была составлена следующая таблица тестов (табл. 4.1). Она содержит входные данные, ожидаемые и реальные результаты работы программы с такими данными:

Таблица 4.1 Таблица тестов для задачи 4

|  |  |  |  |
| --- | --- | --- | --- |
| **Номер теста** | **Входные данные** | **Ожидаемые выходные данные** | **Примечание** |
| 1 | 0,001 | 2,17455941 | 1.1, 2.1 |
| 2 | тест | Введите действительное положительное число. | 1.2, 2.2 |
| 3 | -0,001 | Введите действительное положительное число. | 1.3, 2.2 |

# Задача 5

Даны натуральное число n, целочисленная, квадратная матрица порядка n. Получить b1, …, bn, где bi – это сумма элементов, предшествующих последнему отрицательному элементу i-й строки (если все элементы строки неотрицательны, принять bi = -1).

## Анализ

Для решения задачи нужно пройтись циклом по каждой строке матрицы и считать сумму до нахождения в строке отрицательного числа. После обнаружения записать значение суммы во вспомогательную переменную и суммировать элементы до столкновения со следующим отрицательным значением. Если же отрицательное число не обнаружилось, и строка кончилась, то записать в ответ значение из вспомогательной переменной.

## Проектирование

На основе описанного анализа и алгоритма в нём, была составлена следующая блок схема (см. рис. 5.1).

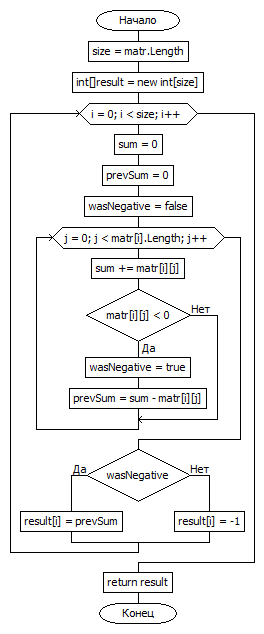


Рисунок 5.1 Блок-схема для задачи 5

## Тестирование

Для тестирования была составлена таблица (см. табл. 5.1), отвечающая критериям чёрного ящика представленного в приложении НОМЕР ПРИЛОЖЕНИЯ .

Таблица 5.1 Таблица тестов для задачи 5

|  |  |  |  |
| --- | --- | --- | --- |
| **Номер теста** | **Входные данные** | **Ожидаемые выходные данные** | **Примечание** |
| 1 | 1, 2, 3 1, 2, 3 1, 2, 3 | -1, -1, -1 | 1.1 |
| 2 | 1, 2, -3 1, -2, 3 -1, 2, 3 | 3, 1, 0 | 1.2, 2.1, 2.2, 2.3 |
| 3 | -1, -2, -3 -1, -3, -3 -1, -4, -3 | -3, -4, -5 | 1.3, 2.1, 2.2, 2.3 |

# Задача 6

Ввести a1, a2, a3, М, N. Построить последовательность чисел ак = ак‑1 + ак‑2  /3 + 3 \* ак‑3. Проверить, существуют ли среди первых N элементов последовательности элементы, равные M, и если существуют, то сколько их. Напечатать N элементов последовательности и номера элементов, равных M.

Для решения задачи использовать рекурсию.

## Анализ

В исходных данных имеются первые три элемента последовательности, следовательно в рекурсивной функции сначала стоит определить условия их вызова. Начиная с 4 элемента формула высчитывания в качестве операндов использует рекурсивную функцию. Для вывода результата необходимо передавать ссылку на массив, имеющий размер равный количеству чисел для вычисления.

Также стоит ограничить ввод количества чисел для вычисления последовательности натуральными числами.

## Проектирование

На основе проведенного анализа была построена блок-схема для рекурсивного метода.

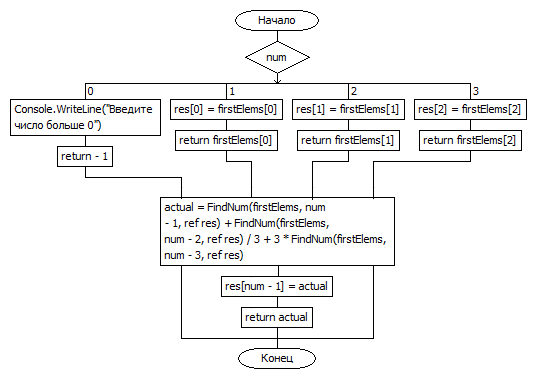
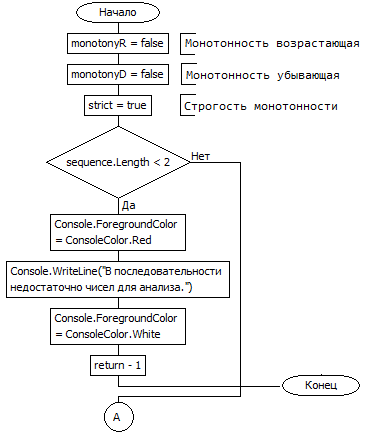
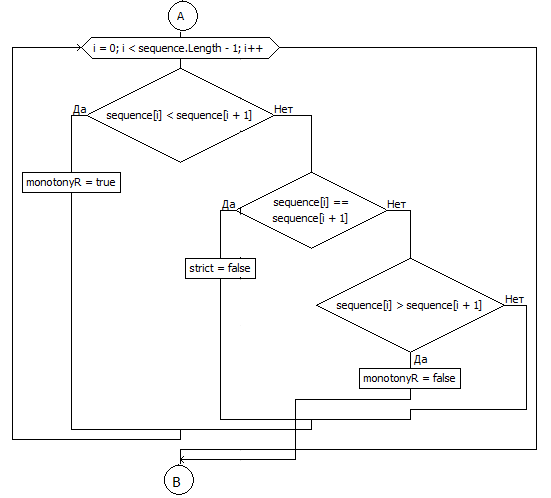
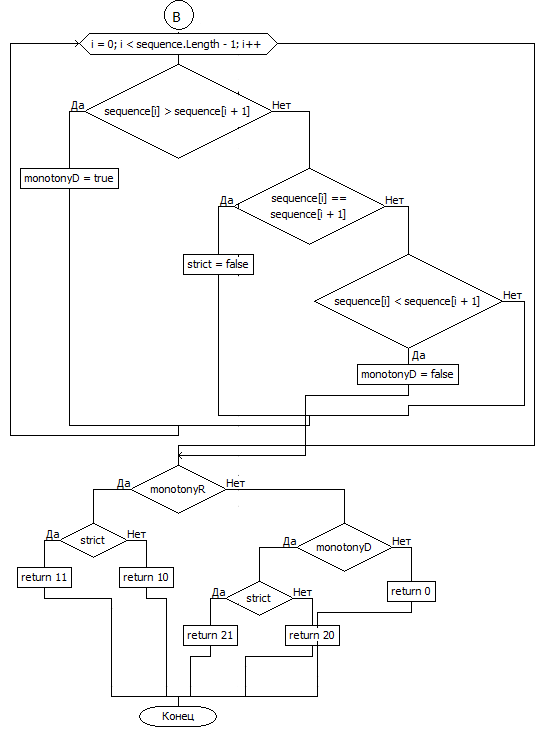


Рисунок 6.1 Блок-схема для рекурсивного метода задачи 6

Дополнительно был разработан метод для анализирования полученной последовательности на монотонность.



## Тестирование

Для тестирования была составлена следующая таблица тестов (табл. 6.1). Она содержит входные данные, ожидаемые и реальные результаты работы программы с такими данными. Таблица покрывает критерии чёрного ящика, представленного в приложении НАЗВАНИЕ ПРИЛОЖЕНИЯ:

Таблица 6.1 Таблица тестов для задачи 6

|  |  |  |  |
| --- | --- | --- | --- |
| **Номер теста** | **Входные данные** | **Ожидаемые выходные данные** | **Примечание** |
| 1 | Изнач. элементы: 1 2 3 Количество вычисл. значений: 0 | "Введите целое число больше 0" | 1.1, 4.2 |
| 2 | Изнач. элементы: 1 2 3 Количество вычисл. значений: 0 Число M: 1 | Последовательность: 1 Номера элементов, которые равны M: 1 Количество чисел равных M равно: 1 | 1.2, 2.1, 3.2, 4.1 |
| 3 | Изнач. элементы: -1 -1 -4 Количество вычисл. значений: 5 Число M: -1 | Последовательность: -1 -1 -4 -7,33333333333333 -11,6666666666667 Номера элементов, которые равны M: 1 2 Количество чисел равных M равно: 2 | 1.3, 2.3, 3.3, 4.1 |
| 4 | Изнач. элементы: -1 2 3 Количество вычисл. значений: 5 Число M: 0 | Последовательность: -1 2 3 0,666666666666667 7,66666666666667 Номера элементов, которые равны M:  Чисел равных M нет. | 1.3, 2.2, 3.1, 4.1, 4.3 |

Также была составлена таблица тестов (см. табл. 6.2) для дополнительной функции – анализа полученной последовательности на монотонность. Критерии чёрного ящика представлены в приложении НАЗВАНИЕ ПРИЛОЖЕНИЯ:

Таблица 6.2 Таблица тестов для доп. функции задачи 6

|  |  |  |  |
| --- | --- | --- | --- |
| **Номер теста** | **Входные данные** | **Ожидаемые выходные данные** | **Примечание** |
| 1 | 1, 2, 3, 6.66, 13.66 | Числовая последовательность монотонная строго возрастающая | 1.2 |
| 2 | 1, -2, 3, 5.3, 0.33 | Числовая последовательность не монотонная. | 1.1 |
| 3 | 1, 1, 2, 5.3, 9 | Числовая последовательность монотонная нестрого возрастающая | 1.4 |
| 4 | -1, -1, -3, -6.33, -10,33 | Числовая последовательность монотонная нестрого убывающая. | 1.5 |
| 5 | -1, -2, -3, -6,66, -13,66 | Числовая последовательность монотонная строго убывающая. | 1.3 |

# Задача 7

Построить суффиксный троичный код с заданными длинами кодовых слов. Кодовые слова выписать в лексикографическом порядке.

## Анализ

Троичный суффиксный код – это способ представления данных в виде комбинации трёх литер, в котором ни одно кодовое слово не является окончанием другого кодового слова. Перед тем как начать вычисление кодовых слов, следует проверить длины на неравенство Макмиллана (см. формулу 7.1).

(7.1)

Где, m – это количество длин слов,

q – это число знаков кодирующего алфавита (в случае данной задачи равно 3),

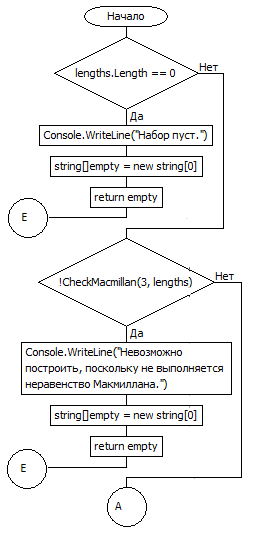
Li – это длины слов.

Если длины не проходят неравенство Макмиллана, то программе необходимо прекратить выполнение.

Для вычисления кодовых слов необходимо воспользоваться методом построения дерева. Но стоит учесть, что в этом варианте будет создаваться префиксная система, значит после получения кодовых слов их требуется инвертировать. Чтобы построить дерево, сначала нужно отсортировать длины слов по убыванию, пройти по каждой из них, подсчитать количество повторений длин в массиве длин слова, а затем пройти по каждой букве кодирующего алфавита и начать строить кодовое слово по длине минус один. После, в зависимости от количества повторений, перебирается последняя буква кодового слова (именно поэтому в предыдущем шаге использовалась длина минус один), затем алгоритм переходит к другим длинам и подобным образом строит слова.

## Проектирование

На основе вышеописанного алгоритма была спроектирована программа, блок-схема представлена на следующих рисунке 7.1.



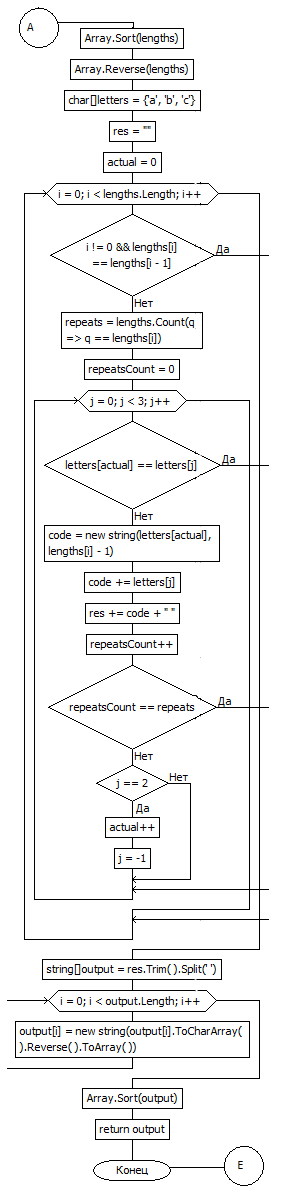


Рисунок 7.1 Блок-схема для задачи 7

Ниже (см. рис. 7.2) представлена блок-схема для доп. метода для вычисления неравенства Макмиллана:

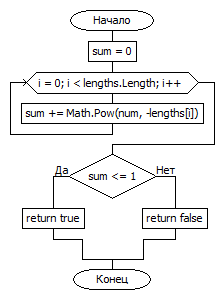


Рисунок 7.2 Блок-схема для метода для вычисления на неравенство Макмиллана

## Тестирование

Таблица 7.1 Таблица тестов для задачи 7

|  |  |  |  |
| --- | --- | --- | --- |
| **Номер теста** | **Входные данные** | **Ожидаемые выходные данные** | **Примечание** |
| 1 | Количество длин кодовых слов: 0 | "Введите целое число больше 0" | 1.1, 4.3 |
| 2 | Количество длин кодовых слов: 1 Длины слов: 3 | baa | 1.2, 2.1, 3.1, 4.1 |
| 3 | Количество длин кодовых слов: 4 Длины слов: 3, 2, 2, 2 | ab ba baa ca | 1.3, 2.3. 3.1, 4.1 |
| 4 | Количество длин кодовых слов: 4 Длины слов: 1, 1, 1, 1 | Невозможно построить, поскольку не выполняется неравенство Макмиллана. | 1.3, 2.3. 3.2, 4.2 |
| 5 | Количество длин кодовых слов: 2 Длины слов: 1, 1 | b c | 1.3, 2.2, 3.2, 4.2 |

# Задача 8

Граф задан матрицей смежности. Найти в нем какой-либо пустой подграф из K вершин. Разработать генератор тестов.

## Анализ

В первую очередь требуется считать матрицу из файла, при этом следует обработать ситуации при вводе нецелых чисел, нескольких пробелов и букв. Затем необходимо найти все внутренние независимые множества. Множество называется независимым, если никакие две вершины этого множества не соединены ребром.

После поиска всех внутренних независимых множеств надо пройти по всем остальным точкам и проверить на наличие общих ребер, это нужно для поиска пустого подграфа. Пустой подграф – подграф, при добавлении к которому хотя бы одной вершины образуется хотя бы одно ребро. Если ребро обнаруживается, то это значит, что внутренняя независимость нарушается, а значит множество является пустым подграфом.

Для генератора тестов должен использоваться датчик случайных чисел для заполнения матрицы смежности.

## Проектирование

Рисунок 8.1 Блок-схема для задачи 8

Проходит по каждой строке матрицы смежности

Проходит по каждому независимому множеству

Проходит по каждой вершине

Вершина имеет общее ребро с вершиной из множества?

Записывает результат

Поиск независимых множеств

Рисунок 8.3 Блок-схема для генератора тестов

Записывает результат

Проверяет вершины в одном множестве на общие смежные вершины

Добавляет все вершины, с которыми вершина не смежна, в доп. массив

Проходит по строкам матрицы

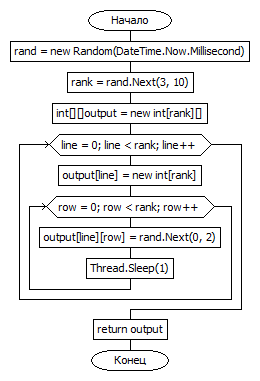


Рисунок 8.2 Блок-схема для метода поиска внутренних независимых множеств

## Тестирование

Для тестирования была составлена следующая таблица тестов (см. табл. 8.1). Представленные тесты покрывают критерии чёрного ящика в приложении НАЗВАНИЕ

Таблица 8.1 Таблица тестов для задачи 8

|  |  |  |  |
| --- | --- | --- | --- |
| **Номер теста** | **Входные данные** | **Ожидаемые выходные данные** | **Примечание** |
| 1 | 0 1 0 1 1 0 0 0 1  1 1 1 0 0 0 0 0 0  1 1 1 1 1 1 1 0 1  0 1 1 1 1 0 0 0 1  1 1 1 0 1 1 0 0 1  0 1 0 1 1 1 1 1 0  0 0 1 0 0 1 1 1 0  1 1 0 1 1 0 0 0 1  1 1 0 1 0 1 0 1 0 | Пустых подграфов нет | 1.1, 3.1 |
| 2 | 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 1 0 2 1 0 1 0 1 0 0 0 0 1 0 1 1 0 1 0 1 0 | Пустые подграфы: 1 3 5 | 1.2, 2.1, 2.3, 3.2 |
| 3 | 0 1 0 1 0 1 1 0 1 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 1 0 1 1 0 1 0 1 0 | Пустые подграфы: 0 2 4 1 3 5 | 1.3, 2.1, 2.3, 3.2 |
| 4 | 0 1 0 1,2 0 0 1 0 1 | В строке 2 введено не целое число | 2.2, 3.3 |

# Задача 9

В программе построен циклический список. Напишите функцию подсчета разности между суммами всех четных и всех нечетных значений, записанных в информационные поля элементов списка.

## Анализ

Для начала необходимо написать класс циклического списка. Для циклического списка требуется класс элемента – узел, который имеет два поля: следующий элемент в списке и число, которое хранит узел. Для каждого кольцевого списка должен быть головной элемент, от которого остальные будут идти связно. Затем в списке надо реализовать методы для добавления элемента.

Для вычисления разницы сначала нужно найти все чётные и нечётные числа. Для этого нужно запустить цикл с условием остановки обнаружение объекта без ссылки на следующий элемент. Затем при обнаружении чётных чисел и нечётных соответственно прибавлять их к вспомогательным переменным. После остановки – вычислить разницу между ними.

## Проектирование

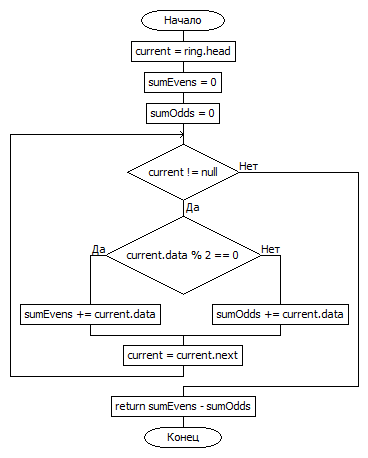


Рисунок 9.1 Блок-схема для вычисления разницы между чётными и нечётными числами в связном списке

На основе вышеописанного алгоритма, представлена блок-схема для вычисления разницы между чётными и нечётными числами в связном списке (рис. 9.1).

Методы класса связного списка (RingList) представлены в таблице 9.1.

Таблица 9.1 Таблица методов класса связного списка

|  |  |  |
| --- | --- | --- |
| № | Метод | Описание |
| 1 | void AddFirst(int data) | Добавляет элемент в начало списка (делает его головным) |
| 2 | void AddLast(int data) | Добавляет элемент после всех элементов |
| 3 | void showNodes() | Выводит все элементы кольцевого списка |

Блок-схемы приведенных методов представлены ниже:

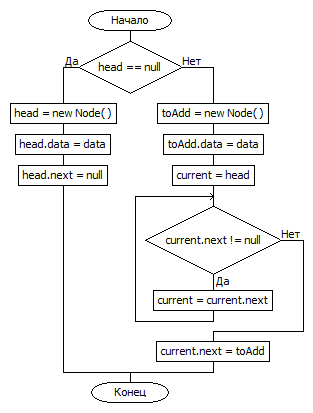
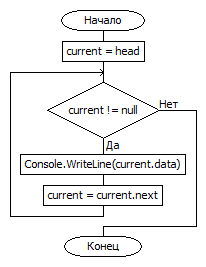
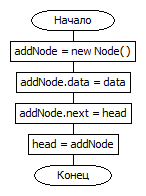


Рисунок 9.4 Блок-схема добавления элемента в конец списка

Рисунок 9.3 Блок-схема вывода элементов списка

Рисунок 9.2 Блок-схема добавления элемента в начало списка

Дополнительный рекурсивный метод для вычисления разницы между суммами чётных и нечётных чисел в кольцевом списке представлен на рис. 9.5.

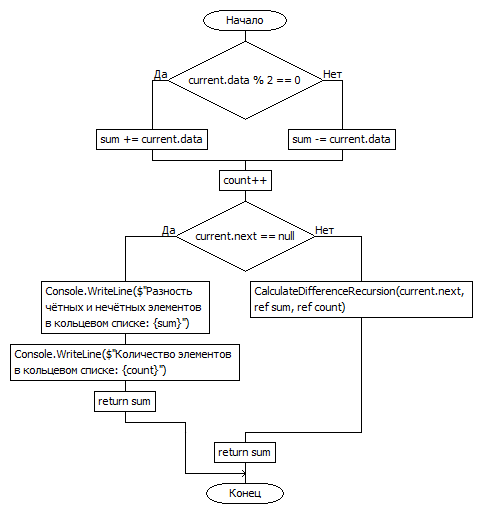


Рисунок 9.5 Блок-схема для вычисления разницы между чётными и нечётными числами в связном списке рекурсивным методом

## Тестирование

Для тестирования была составлена следующая таблица тестов (см. табл. 9.2). Она содержит входные данные, ожидаемые и реальные результаты работы программы с такими данными. Критерии чёрного ящика приведены в приложении НАЗВАНИЕ ПРИЛОЖЕНИЯ

Таблица 9.2 Таблица тестов для задачи 9

|  |  |  |  |
| --- | --- | --- | --- |
| **Номер теста** | **Входные данные** | **Ожидаемые выходные данные** | **Примечание** |
| 1 |  | Введите целые числа | 1.1, 3.1 |
| 2 | 1 | -1 | 1.2, 2.2, 3.2 |
| 3 | 2 2 2 | 6 | 1.3, 2.1, 3.2 |
| 4 | 2 3 4 | 3 | 1.3, 2.3, 3.2 |
| 5 | 0 0 0 | 0 | 1.3, 2.4, 3.2 |
| 6 | 1.2 2 3 4 5 | Введите целые числа | 1.3, 2.5, 3.1 |

# Задача 10

Написать метод уничтожения дерева.

## Анализ

Задача подразумевает собой написание дерева. Для создания дерева сначала необходимо определить класс узла дерева, который имел бы поля: данные, ссылка на левый элемент, ссылка на правый элемент. Само дерево должно иметь корень. Дерево должно иметь методы для создания дерева – этот метод можно реализовать рекурсивный методом. Так как все ветви дерева идут от корня, значит уничтожить дерево – это удалить ссылку на другие ветви у корня.

## Проектировние

Публичные методы класса дерева (Tree) представлены в таблице 10.1.

Таблица 9.1 Таблица методов класса дерева

|  |  |  |
| --- | --- | --- |
| № | Метод | Описание |
| 1 | public void DeleteTree() | Уничтожает дерево |
| 2 | private TreeElement<T> MakeTree(int num, TreeElement<T> root, ref int i, params T[] datas) | Метод для рекурсивного заполнения дерева. |
| 3 | public void Maximum() | Дополнительный метод для получения макс. значения из дерева |
| 4 | private TreeElement<T> GetMax(TreeElement<T> root, ref TreeElement<T> max) | Дополнительный рекурсивный метод, считает макс. на ветви. |
| 5 | void Show() | Вывод дерева в консоль |
| 6 | private void ShowTree(TreeElement<T> root, int l) | Рекурсивный метод для вывода содержимого ветви |

Блок-схемы представлены в приложении НАЗВАНИЕ ПРИЛОЖЕНИЯ

## Тестирование

Таблица тестов и критерии тестирования представлены в приложении НАЗВАНИЕ ПРИЛОЖЕНИЯ

# Задача 11

«Исправление ошибок». Пусть по некоторому каналу связи передается сообщение, имеющее вид .последовательности нулей и единиц (или, аналогично, точек и тире). Из-за помех возможен ошибочный прием некоторых сигналов: нуль может быть воспринят как единица и наоборот. Можно передавать каждый сигнал трижды, заменяя, например, последовательность 1, 0, 1 последовательностью 1.1,1,0,0,0,1,1,1. Три последовательные цифры при расшифровке заменяются той цифрой, которая встречается среди них по крайней мере дважды. Такое утраивание сигналов существенно повышает вероятность правильного приема сообщения.

## Анализ

Для начала нужно проверить количество чисел в последовательности, если не кратно трём, то вывести предупреждение. Иначе, необходимо запустить цикл по всем цифрам последовательности. Затем каждые три числа суммируются во вспомогательную переменную и после третьего элемента идёт проверка суммы. Если сумма меньше 2, то записывает в ответ нуль, иначе единицу.

## Проектирование

На основе вышеописанного анализа и приведенного алгоритма составлена блок-схема:

Рисунок 11.1 Блок-схема для задачи 11

Да

Да

Нет

Нет

Записывает в результат 0

Записывает в результат 1

Сумма больше 2?

Три цифры пройдено?

Прибавляет цифру к сумме

Проходит по всем цифрам последовательности

Проверка количества чисел в последовательности

Во время разработки были написаны дополнительные методы для конвертирования полученного числа в десятичную и шестнадцатеричную системы счисления.

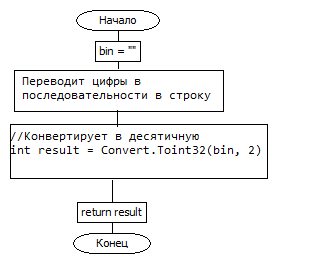
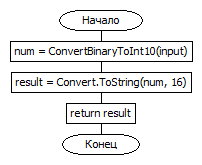


Рисунок 11.2 Блок-схема для доп. метода для перевода полученного результата в 10-ричную систему

Рисунок 11.3 Блок-схема для доп. метода для перевода полученного результата в 16-ричную систему

## Тестирование

Для тестирования была составлена следующая таблица тестов (табл. 11.1). Она содержит входные данные, ожидаемые и реальные результаты работы программы с такими данными. Критерии по чёрному ящику представлены в приложении НАЗВАНИЕ ПРИЛОЖЕНИЯ.

|  |  |  |  |
| --- | --- | --- | --- |
| **Номер теста** | **Входные данные** | **Ожидаемые выходные данные** | **Примечание** |
| 1 | 1 1 1 1 0 1 0 1 0 | 1 1 0 | 1.1, 1.2,2.2, 2.3, 3.3, 4.1 |
| 2 |  | Ваш массив пуст | 3.1, 4.2 |
| 3 | 1 | Исключение: в последовательности недостает чисел | 3.2, 4.3 |
| 4 | 1 0 1 1 0 1 0 1 0 | 1, 1, 0 | 1.1, 2.1, 2.2, 2.3, 3.3, 4.1 |

# Задача 12

Выполнить сравнение двух предложенных методов сортировки одномерных массивов, содержащих n элементов, по количеству пересылок и сравнений.

Для этого необходимо выполнить программную реализацию двух методов сортировки, включив в нее подсчет количества пересылок (т.е. перемещений элементов с одного места на другое) и сравнений.

Провести анализ методов сортировки для трех массивов: упорядоченного по возрастанию, упорядоченного по убыванию и неупорядоченного.

Все три массива следует отсортировать обоими методами сортировки.

Найти в литературе теоретические оценки сложности каждого из методов и сравнить их с оценками, полученными на практике.

Сделать выводы о том, насколько отличаются теоретические и практические оценки количества операций, объяснить почему это происходит. Сравнить оценки сложности двух алгоритмов.

Методы сортировки:

1. Сортировка простыми вставками.
2. Сортировка простым выбором.

## Анализ

Сортировка простыми вставками – это метод сортировки, суть которого заключается в просматривании элементов по одному и вставке нового элемента в предшествующий уже упорядоченный набор.

|  |  |  |
| --- | --- | --- |
| Лучшее время | O (n) | 0 пересылок |
| Среднее время | O () | O() пересылок |
| Худшее время | O () | O() пересылок |
| Затраты памяти | O (n) |  |

Сортировка простым выбором – это метод сортировки, суть которого заключается в поиске минимального (максимального) значения в списке и в замене последнего (первого) элемента в неотсортированной части массива до тех пор, пока массив не будет отсортирован полностью.

|  |  |  |
| --- | --- | --- |
| Лучшее время | O () | 0 пересылок |
| Среднее время | O () | O() пересылок |
| Худшее время | O () | O() пересылок |
| Затраты памяти | O (n) |  |

Для подсчёта количества сравнений и пересылок следует использовать дополнительные переменные, которые требуется инкрементировать в соответствующих частях программы.

## Проектирование

Блок-схемы алгоритмов сортировки вставками и сортировки выбором соответственно представлены на рисунках 12.1 и 12.2.

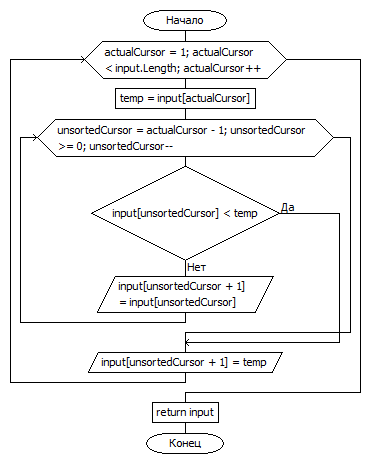
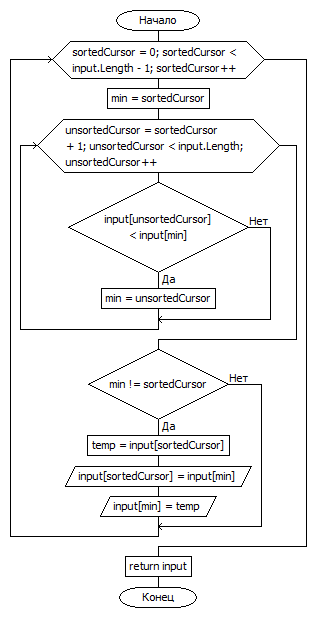


Рисунок 12.2 Блок-схема для алгоритма сортировки выбором

Рисунок 12.1 Блок-схема для алгоритма сортировки вставками

Также был дополнительно реализован алгоритм сортировки подсчётом.

Сортировка подсчётом – это алгоритм, суть которого заключается в подсчёте количества вхождений каждого значения в массиве и заполнения массива подсчитанными элементами в соответствующих количествах.

|  |  |  |
| --- | --- | --- |
| Лучшее время | O (n) | 0 пересылок |
| Среднее время | O (n + k) | O(n) пересылок |
| Худшее время | O (n + k) | O(n) пересылок |
| Затраты памяти | O (n) |  |

Где k – это разность между максимальным и минимальным элементом массива.

Блок схема алгоритма представлена на рисунке 12.3.

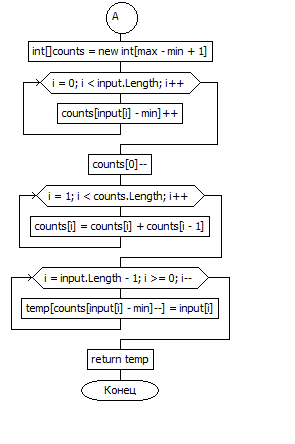
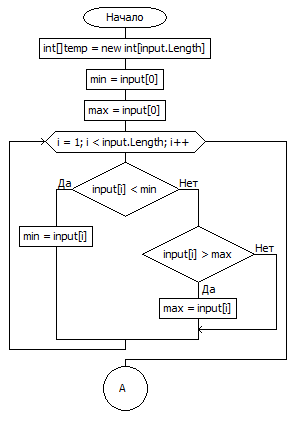


Рисунок 12.3 Блок-схема для алгоритма сортировки подсчётом

## Тестирование

Для тестирования была составлена представленная ниже таблица (см. табл. 12.1). Данные тесты покрывают критерии чёрного ящика, представленные в приложении НАЗВАНИЕ ПРИЛОЖЕНИЯ

Таблица 12.1 Таблица тестов для задачи 12

|  |  |  |  |
| --- | --- | --- | --- |
| **Номер теста** | **Входные данные** | **Ожидаемые выходные данные** | **Примечание** |
| 1 | 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 | 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 | 1.1, 2.2, 3.1, 3.2, 3.3 |
| 2 | -10, -9, -8, -7, -6, -5, -4, -3, -2, -1 | -1, -2, -3, -4, -5, -6, -7, -8, -9, -10 | 1.2, 2.3, 3.1, 3.2, 3.3 |
| 3 | 1, -2, 3, 4, 5, -7, 12, 0, 13 | -7, -2, 0, 1, 3, 4, 5, 12, 13 | 1.3, 2.1, 3.1, 3.2, 3.3 |

## Сравнение алгоритмов сортировки

Практические данные о количестве сортировок и пересылок представлены в таблице 12.2.

Таблица 12.2 Таблица сравнений алгоритмов

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Вставками | | Выбором | | Подсчётом | |
| Сравнений | Пересылок | Сравнений | Пересылок | Сравнений | Пересылок |
| Упорядоченный по возрастанию массив | 27 | 9 | 108 | 9 | 56 | 19 |
| Упорядоченный по убыванию массив | 99 | 54 | 108 | 9 | 47 | 19 |
| Неупорядоченный массив | 61 | 27 | 108 | 9 | 47 | 19 |

Практические результаты подсчёта сравнений и пересылок совпадают с теоретическими. Если смотреть на данные о сортировке подсчётом, то может показаться, что она самая сбалансированная среди прочих, но стоит учитывать особенности алгоритма. Чем больше сами числа в массиве, тем больше сравнений и пересылок произведет алгоритм. В тестирующем наборе данных самым большим числом являлось 10. Но если массив содержит, к примеру, число 127, то уже минимальное количество пересылок и сравнений соответственно будет являться 164 и 136. Это означает, что рассматриваемый алгоритм уместно использовать над массивом данных, имеющим небольшие значения.

Сортировка вставками является одним из наименее ресурсоёмких алгоритмов среди рассматриваемых, но только на упорядоченном по возрастанию массиве. Можно заметить, как у алгоритма резко увеличилось количество пересылок и сравнений на массиве с убывающими значениями. В сравнении с ней алгоритм сортировки выбором является крайне предсказуемым и при этом показывает наименьшее количество пересылок среди всех прочих алгоритмов.

Теоретические данные наводят на мысли о том, что нет абсолютного «победителя» алгоритма и к его подбору при использовании в различных системах требуется обращать внимание на «внутреннее» устройство алгоритма, знать его сильные и слабые стороны.